Kaimi Huang (kh2908)

Task 1

1. The issues with RNN are linear interaction distance and lack of parallelizability. With linear interaction among a sequence of words, it's hard to learn long-distance dependencies due to vanishing gradient. And lack of parallelizability means future RNN hidden states cannot be computed before past RNN hidden states have been computed. This feature inhibits training on large datasets. The building block of self-attention in Transformer solves these issues.

2. Tokenizing the corpus by individual words preserves semantic meaning and allows the model's predictions and embeddings to be interpretable. The downside is the vocabulary size can be large, leading to long training time and high memory requirements. Also, rare words or misspelled words may be represented as OOV. Using OOV results in loss of information. On the other hand, tokenizing by characters handles unseen words and misspellings well. The vocabulary size is much smaller, which means shorter training time and less memory usage. However, semantic meaning that is present at word level would be lost. Also, sequence length can be much longer, increasing computational requirements.

3. In the encoder-decoder transformer language model, inputs are processed in the encoder. The encoder uses self-attention to represent the relations among all tokens in the input sequence and creates a context vector to capture the sequence's meaning. In other words, the encoder is tasked to understand and extract important information from the input. The decoder then performs cross-attention to decode the context vector and generate the output sequence word by word. Relying on a combination of the context vector and the word it just generated, the decoder autoregressively produces the rest of the sequence.

   In contrast, the autoregressive transformer language model has only a decoder that both encodes the prompt and generates the output. It does so by using masked self-attentions to capture the relations among input tokens. The masked self-attention values for each word contain information from all the other words that came earlier. This helps give each word context and establishes how each word in the input prompt is related to the others. During language generation, the model uses past outputs as input to decide the most probable next word.

4. When training an encoder-decoder model for machines translation, I would provide a parallel corpus. For instance,
   Input: "The bunny is cute."
   Target: "el conejito es lindo."

   When training an autoregressive transformer model, the training data would be:
   "'The bunny is cute' in Spanish is 'el conejito es lindo.'"

   During evaluation of the decoder model, I would provide the following prompt and expect the model to generate the translation:
   "What is "the elephant is cute" in Spanish?"

5. Yes, since autoregressive models generate sequences one word at a time, predicting the next token based on the preceding ones, if the training data is in the structure of a sequential, running text, then it is possible.

6. In beam search, instead of choosing the best token to generate at each timestep, we keep k possible tokens at each step. At the first step, the k best tokens are the hypotheses, or output-sequence-so-far. Then, they are extended by their next best tokens. Each hypothesis has k extensions, but only the best k hypotheses are kept. This process continues until the end-of-sentence tokens are met for k hypotheses. Therefore, unlike greedy decoding, the final output sequence is decided not by concatenating the tokens with the highest probabilities at each t but is the globally most probable sequence. Greedy decoding is preferred in situations when computational resource is limited.

7. Under the premise that there is a large amount of fine-tuning data, the transformer would perform equally well. Because the <JUNK> tokens do not have meaning, the self-attentions in the encoder would assign very little importance to these tokens, leaving high importance to the essential parts for translation outputs. However, if the size of training data is small, inserting <JUNK> tokens in the data would improve the model's generalization.

Task 2

1. Preparation
2. Obtain Internal States
3. Train a Classifier

- How large did you choose your train/validation/testing sets to be?

  The train/val/test datasets have an 6:2:2 ratio. That means the training set has 300 samples, validation set has 100, and the test set has 100.

- If you have multiple layers in your classifier that reduce the input's dimensionality as it goes through the classifier, why did you choose these layer sizes?

  The first layer uses the MaxPool2d function to reduce the tensors' dimension from [len_of_2nd_sentence, 768] to [1, 768]. Doing so retains only the maximum self-attention representation among all tokens for each hidden state. Then, the tensors are flattened to [768]. The subsequent fully connected layer reduces the number of neurons from 768 to 128. The last fully connected layer reduces the neuron number from 128 to 3, which is the number of classes.

- What steps did you take to improve your classifier's generalization?

  To improve the model's generalization, I tried three methods:
  1. In the intermediate fully connected layer, I tried adjusting the number of hidden units. The numbers experimented were 128, 64, 32, 16. But for this classifier, it seems that the number of parameters did not affect generalization; all the experiments allowed the classifier to converge.
  2. I implemented L2 regularization and tried several lambdas. Lambda of 0.09 has been tested to have the lowest validation loss.
  3. In the network, I tried adding a dropout layer and experimented various dropout rates. However, this method did not improve generalization. Therefore, it was not implemented.

- What metric do you choose to have as your final performance evaluation of your classifier; why did you choose this metric?

  I used accuracy – (Number of Correct Predictions) / (Total Number of Predictions). Because the dataset is balanced (see entailment distribution of the dataset below), using accuracy is suitable and easy to interpret.

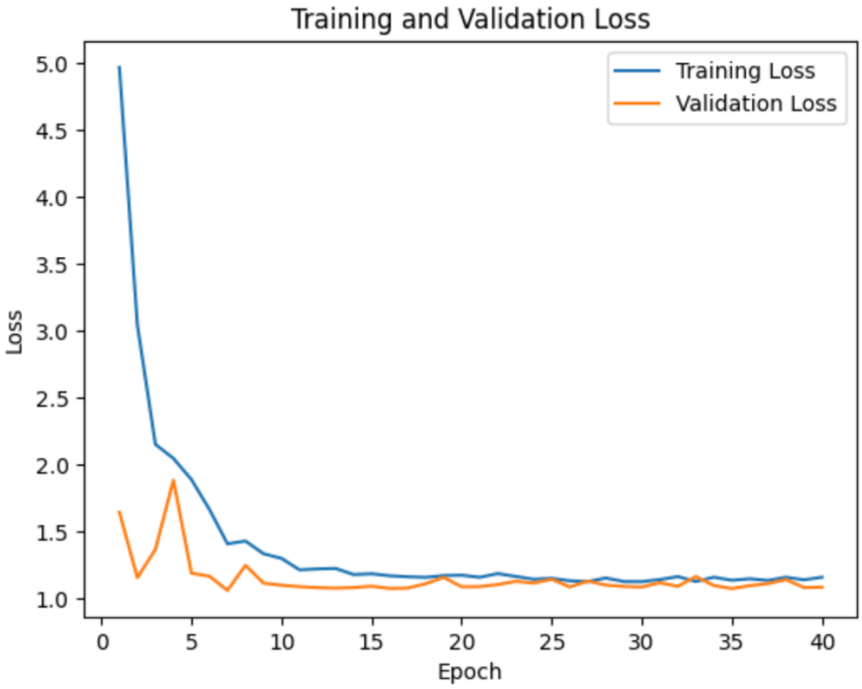
```
Out[47]: {'contradiction': 168, 'neutral': 177, 'entailment': 155}

In [48]: [count/500 for entailment, count in class_count.items()]
Out[48]: [0.336, 0.354, 0.31]
```

- Display your loss curves during training (for train and validation loss), and the final test set metric value your classifier achieved. Reflect on your classifier's performance: did it do well? Did it do horribly? What does this tell you about the saliency of entailment relations (i.e., does GPT2

"know" that it just said a contradiction or an entailment?) in the layer you just probed (0th layer MLP)?

The final test set accuracy was 40%. Based on this value alone, the classifier did better than randomly guessing – 33.33%. This means GPT2 might know that it just said a contradiction or an entailment.



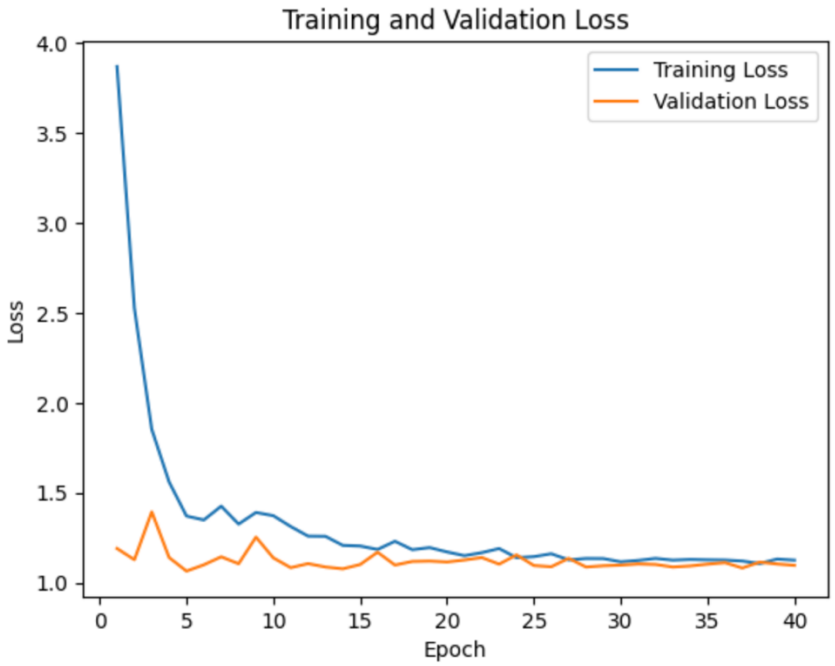

```
Accuracy on test set is: 0.4
```

4. Probe other layers

- Report your training curves and final test set performances. Given your results, what can you conclude about the "signal" of entailment/contradiction in these deeper layers, compared to layer 0 of the transformer? Do you notice any trends between the depth of the layer you probe and your test set performance?

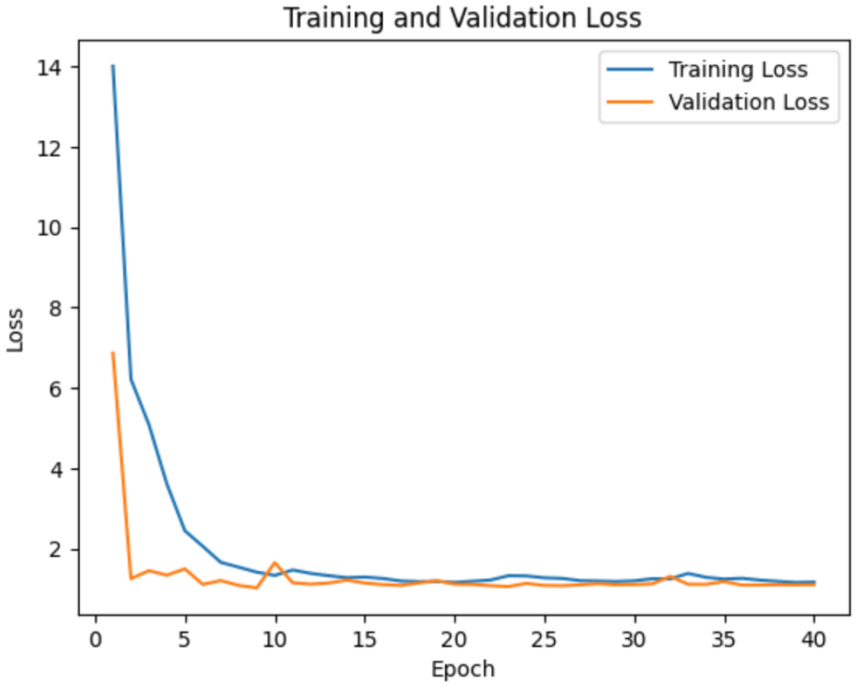
  I probed two other MLP layers – the 5th and the 10th. The classifier had 38% accuracy on the 5.mlp test set and 32% accuracy on the 10.mlp test set. The trend shows the deeper the layer, the worse the test set performance became. Based on the results, it seems that the "signal" of entailment/contradiction is the strongest in the early layers of GPT2 and it lessens in deeper layers.

**Training Curves of the 5.mlp layer:**




```
Accuracy on test set is: 0.38
```

**Training Curves of the 10.mlp layer:**




```
Accuracy on test set is: 0.32
```

5. Bonus Task 1

- What if we used a different prompt? Try something different than “This has the same meaning as”. Compare the performance of your probe when it was trained on the prompt we used before, compared to whatever new prompt you conjure up. Do you see any differences in test set performance? Why might this be the case?

  The new prompt I used was “<sentence1> The previous sentence does not provide enough evidence for the following sentence: <sentence2>.” The idea behind choosing this prompt is to manipulate GPT2’s likelihood to come up with the designated sentence2s.

  With the new prompt, GPT2 would be very likely to generate contradictory text as sentence2, less likely but still likely to generate neutral text, and unlikely to generate text that entails.

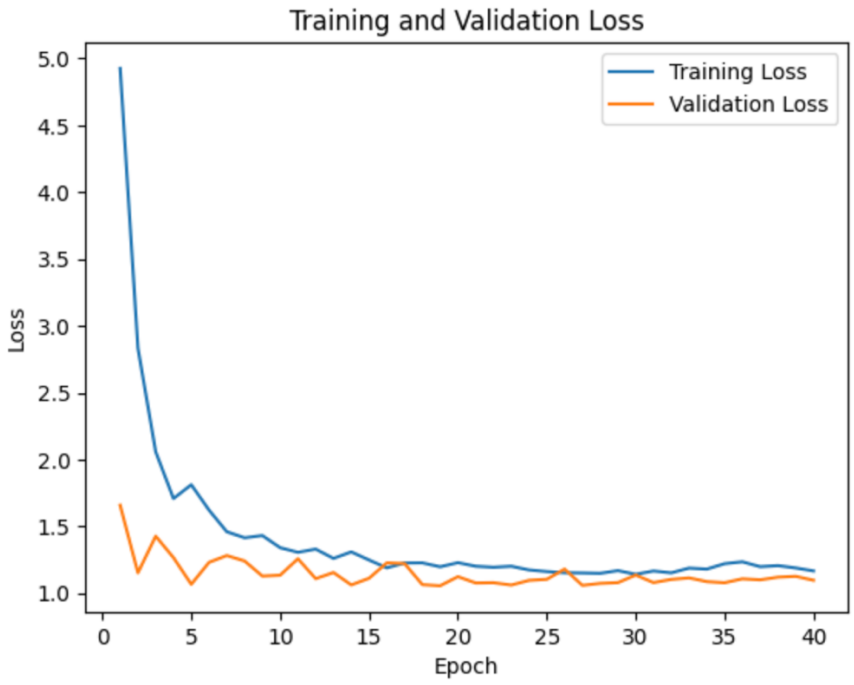
  With the original prompt, GPT2 would be unlikely to generate contradictory text as sentence2, likely but to a low degree to generate neutral text, and very likely to generate text that entails as sentence2.

  As compared above, the degree of likelihood for the neutral group is stronger, more distinguishable with the new prompt. And if the likelihood information is reflected in its hidden states, then the pattern would be more prominent with the new prompt, allowing my classifier to categorize the extracted hidden states with higher accuracy.

  Results show that with the new prompt, the trend of decreasing “signal” remains, the first layer is 5% higher in accuracy, while the other layers’ remain the same. The results suggest the hypothesis above may be correct and GPT2 may know it just said something contradictory.

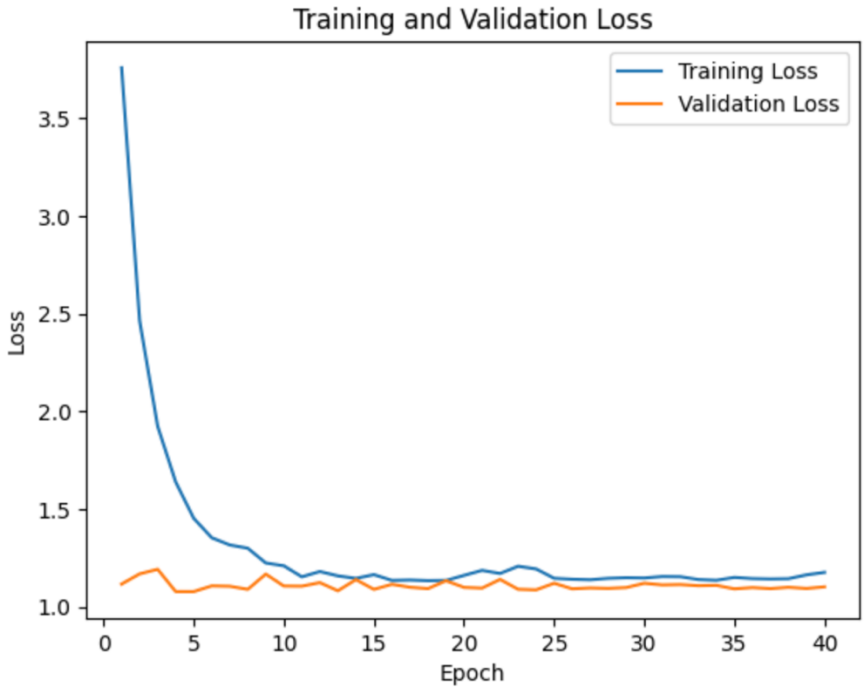
  Accuracy on 0.mlp test set: 45%
  Accuracy on 5.mlp test set: 38%
  Accuracy on 10.mlp test set: 32%

**Training Curves of the 0.mlp layer:**



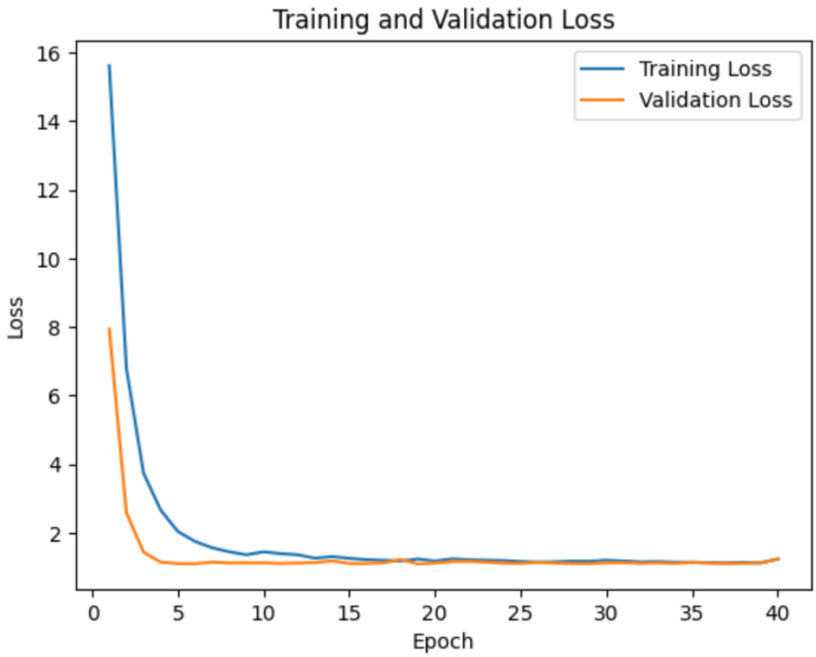

```
Accuracy on test set is: 0.45
```

**Training Curves of the 5.mlp layer:**




```
Accuracy on test set is: 0.38
```

**Training Curves of the 10.mlp layer:**




```
Accuracy on test set is: 0.32
```

6. Bonus Task 2

- Do you think larger models will "know" that they've said something contradictory compared to what they've just said, to a better degree than what you tested here today? Why or why not? Briefly explain.

  Yes, I think so, because larger models had been trained with more data and they have larger capacity to capture more nuances.

7. Bonus Task 3

- Why do you think model internal states might possess the information we just probed for? Think about what models like GPT2 were trained on, and how the features of this data might influence the saliency of such features (if GPT2 knows that it just said something contradictory) in a model's internal states.

  Models like GPT2 were trained on a large corpus of internet text, which includes a diverse range of sources such as articles, books, and websites. Most of these sources do not contain contradictory content. That means a prominent feature of the training data is entailment (rather than contradiction). Such feature is captured in the fixed parameters of the pre-trained models. When given a prompt, the parameters allow the models to generate coherent and contextually relevant text like the training data. During inference, the internal states hold information that represent the prompt and the pretrained knowledge. However, if we force the models to generate contradictory statements, then the model would have to somehow make modification without updating the fixed parameters. And such modification would be captured in the internal states.